Unraveling Timewarp: What all the Fuzz is About?

Sarani Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur,India
sarani.bhattacharya@cse.iitkgp.ernet.in

Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur,India
chester@cse.iitkgp.ernet.in

Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur,India
debdeep@cse.iitkgp.ernet.in

ABSTRACT

Timing attacks are a threat to networked computing systems especially the emerging cloud computing infrastructures. The precision timestamp counters present in modern microprocessors is a popularly used side channel source for timing information. These counters are able to measure the variability of timings that are caused from micro-architectural effects, like cache access patterns and branch miss predictions, and have been routinely used for demonstrating practical attacks against well known ciphers. Recently, researchers have attempted to inhibit precision timing measurements by fuzzing the timestamp, through a time-warped mechanism. In this paper, we demonstrate that in spite of fuzzing time, timing attack are still possible.

Keywords : time-fuzzying, Timewarp, timing attack, cache attacks.

1. INTRODUCTION

Over the next decade, as cloud computing becomes a commodity, Cloud Service Providers (CSP) will compete on service quality and price. In order to increase the percentage of operations that will be moved from in-house IT to cloud service providers and their data centers, cloud infrastructures will need to ensure the security of a customer's applications and the integrity of its data from other applications, which are returned to the cloud. A serious concern to apply mathematically strong ciphers for security is from the threat of side channel attacks. These attack techniques target the underlying implementation to obtain knowledge of the secret key. One classic form of a side-channel is the timing information, which is revealed by standard computing platforms. This can potentially be used for mounting attacks [14] on cipher implementations executing on remote or local systems.

Most implementations of symmetric key block ciphers use table lookups for acceleration of encryption. These table look ups, can be imagined as data arrays, whose elements indicate the round transformations of the block cipher. The data arrays are randomly accessed based on the plaintext and the key, which are inputs to the cipher. During these accesses, there are scenarios when the data array is being accessed at same locations, resulting in cache hits. In other scenarios we call it a cache miss. The timing of an encryption thus varies depending on these hit-miss patterns. By making several timing measurements, an attacker determines the unknown key. Similarly, if the attacker fills the data cache with known data and then allows the cipher to execute, after execution of the cipher, the attacker can again access the elements with which filled the cache, the elements that result in a miss are the ones used by the cipher. Since the cache miss time is significantly greater then the cache hit time, the attacker successfully finds a match of old contents of the cache with the present cache contents and conclude which of the elements are being used by the cipher.

Although there are several timing attacks that are possible such as [2, 3, 13, 14, 19, 20], we focus on attacks that use symmetric key block ciphers and the variations in timing caused by cache memories. These attacks are called cache-timing attacks. However, the results in the paper can be easily adapted for other ciphers and attack techniques. Cache timing attacks utilize the fact that a cache hit takes much lesser time compared to a cache miss. There are two forms of cache timing attacks: access-driven and time-driven. Access driven attacks [11, 16, 18, 25] utilize a spy process to obtain the cache access patterns of the cipher implementation, while timing attacks such as [4, 5, 7, 8, 26, 27] monitor the encryption time and utilize statistical means to predict the secret information.

Several technologies exist to counter cache attacks. Depending on where they are applied, the countermeasures can be broadly classified into two categories: software and hardware technologies. Technologies for software include techniques such as bitslicing, preloading of tables into the cache, randomization, etc.. All these methodologies with the exception of bitslicing [6], suffer from a degradation in the performance. They are therefore not usable in cloud computing infrastructures, where performance is an important concern. The alternative methods are based on hardware modifications, have comparatively less overhead on the performance, and in general offer more security. Some of the well known methods are as follows:

- *Time Fuzzing*: Randomizing time by adding a random offset to timestamp counters. This countermea-

sure can be overcome by taking more number of measurements, thus averaging out the randomness added to the time [11, 15, 20].

- *Specialized Instructions*: for example AES-NI [24]. They are however only applicable for AES, and not for other ciphers.
- *Special Cache Designs*: such as RPCache [28] and Non-monopolizable cache memories [10]. These techniques require some portion of the cache memory to be configured in a way which would prevent cache attacks. They are only applicable to access-driven attacks and not timing attacks.

Recently a time-fuzzing countermeasure for timing attacks has been proposed called Timewarp [15]. This technique is based on the fact that most timing attacks make use of the RDTSC (ReaD TimeStamp Counter) instruction to measure accurately the timing of various operations. This instruction is presently available in the user mode and does not require supervisor privilege, thus making them a tool for attacker's with only user level privileges. However escalating the privilege of RDTSC instruction to privilege mode cannot be done now, as several benign applications use this instruction. As an interesting countermeasure, Timewarp measurements thwart attacks by limiting the fidelity of fine-grained timing measurements and performance counters. Thus the attacker finds it difficult to distinguish between various micro-architectural events. Timewarp measurements are shown to resist both forms of cache-timing attacks since it fuzzies the time within an interval (referred as the epoch). In this paper, we show that the attacker can still mount these cache-timing attacks even when the RDTSC timing is fuzzied with the Timewarp countermeasure. We justify this with experimental validations on Intel Xeon E5410 and Intel Xeon E5606 machines.

The organization of the paper is as follows:- The following section provides a brief idea of the existing time fuzzing schemes and their defects and then explains the Timewarp mechanism. In Section 3 we mathematically analyze the properties of the Timewarp. On basis of this analysis, in Section 4, we first present an overview of an attack strategy based on the Timewarp time, and in the subsections 4.2 and 4.3 we provide the attack strategies for access-driven and time-driven attacks in detail. Section 5 provides the experimental validations for the attack strategy and the final section contains conclusion of the work we present here.

2. PRELIMINARIES

In this section we first give a brief introduction to time-fuzzing schemes for Timewarp algorithm.

2.1 Time fuzzing

The RDTSC (ReaD Timestamp Counter) instruction returns the current value of the timestamp counter. During execution of an operation if we insert two RDTSC instructions, one before and the other one just after the operation of the cipher then we can obtain the execution time of the operation accurately. Existing techniques for thwarting attacks that make use of the timestamp counters include

- Disabling RDTSC instructions [20]: Disallowing any RDTSC instruction from the user space was considered as a simple and effective way of preventing the attacks.
- Masking the least significant bits of RDTSC [18]: This method obscured RDTSC by reducing its precision.
- Adding random offset to RDTSC [12]: In this technique a random value was added to the timestamp.

Each of these alternatives have serious problems. Disabling RDTSC may hamper the functioning of the systems, while masking bit techniques can be attacked by statistical analysis and random offset technique has the serious problem of time going backwards. In ISCA'12 [15] a modification to RDTSC instruction is introduced called Timewarp. The modified RDTSC (which we denote as twRDTSC) introduces obfuscation to the timestamp measurements and thus fuzzies the time. The method of obfuscation requires the time to be divided into epochs (E). Epochs may vary in length randomly from 2^{e-1} to $2^e - 1$ where e is referred to as the level of obfuscation and is set by the Operating System.

When an RDTSC instruction is observed during an encryption, the subsequent epoch boundary is determined, the instruction is stalled until the end of the current epoch for D_{r_1} cycles and the timestamp counter is read on the Epoch boundary. When the epoch boundary is reached, the instruction is subjected to another stall for random number of cycles in range $[0, E)$ denoted as D_{r_2}. The sum of two consecutive stalls $D_{r_1} + D_{r_2}$ is termed as the real offset(D_R). Then a random value in $[0, E)$ is added to the timestamp to further fuzzy it, this being named as the apparent offset(D_A). For fuzzing two consecutive RDTSC instructions, both lying in the same Epoch, the second RDTSC instruction is moved to the consecutive Epoch to maintain the strictly increasing property of time. The steps of fuzzing RDTSC instruction is given in statements as Algorithm 1.

Algorithm 1: Timewarp Measurement steps

begin
- Choose obfuscation level e that varies between $0 - 15$.
- Time is partitioned into epochs(E), epoch lengths varying randomly from 2^{e-1} to $2^e - 1$ cycles.
- An RDTSC instruction when encountered, is stalled to the end of the current epoch (D_{r_1} clock cycles) and the timestamp counter is read.
- RDTSC instruction is again subjected to a random delay ($D_{r_2} \in [0, E)$) in the subsequent epoch.
- The real offset delays the execution of each RDTSC until a random time in the subsequent epoch i.e., $D_R = D_{r_1} + D_{r_2}$.
- Then a random number of cycles in the range $[0, E)$ termed as the apparent offset (D_A) is added to the timestamp counter.

end

3. ANALYSIS OF TIMEWARP MEASUREMENTS

In this section, we present a mathematical representation of the Timewarp algorithm and an analysis of its statistical behavior. We denote Timewarp timestamp function as $twRDTSC$. Let t be the clock cycle when the $twRDTSC$ instruction is encountered. Ideally a non-fuzzied instruction

should return (approximately) t as the timestamp. However Timewarp fuzzies t as shown below:

1. Wait until the end of the current epoch. The current epoch completes at $T = (\lfloor \frac{t}{E} \rfloor + 1) \times E$.
2. Let $D_{r_1} = T - t$
3. Generate two random number $D_{r_2} \in [0, E)$ and an apparent offset $D_A \in [0, E)$.
4. Add D_A to T and return the result after D_{r_2} clock cycles.

Thus, the fuzzied timestamp is returned in the (approximately) $t + D_{r_1} + D_{r_2}$ clock cycle instead of the clock cycle t. The fuzzied timestamp returned is

$$T = ((\lfloor \frac{t}{E} \rfloor + 1) \times E) + D_A \quad (1)$$

instead of t. According to this analysis, the attacker can never make timing measurements less than E. In the next part of the section, we analyze the statistical properties of Timewarp when multiple measurements are made.

Statistical Analysis of Timewarp : In order to measure the time of an event in a Timewarp fuzzied scheme, the adversary makes two twRDTSC() calls and gets the timestamp difference in return. The pseudo code is shown below.

```
Ts = twRDTSC()
    \*event whose time is to be measured*\
Te = twRDTSC()
T = Te - Ts
return T
```

From Equation 1 T_s can be expressed as,

$$T_s = (\lfloor \frac{t_s}{E} \rfloor + 1) \times E + D_{A_s}, \quad (2)$$

for a random apparent offset D_{A_s} and clock cycle t_s and

$$T_e = (\lfloor \frac{t_e}{E} \rfloor + 1) \times E + D_{A_e}, \quad (3)$$

for another random apparent offset of D_{A_e} and clock cycle t_e. Therefore the fuzzied time of the event $T = T_e - T_s$ can be expressed as follows:

$$T = [(\lfloor \frac{t_e}{E} \rfloor + 1) \times E + D_{A_e}] - [(\lfloor \frac{t_s}{E} \rfloor + 1) \times E + D_{A_s}] \quad (4)$$

The equation can be simplified and expressed as

$$T = (\lfloor \frac{t_e}{E} \rfloor \times E - \lfloor \frac{t_s}{E} \rfloor \times E) + (D_{A_e} - D_{A_s}). \quad (5)$$

Assume that the apparent delays D_{A_s} and D_{A_e} are uniformly distributed in the range $[0, E)$. Therefore, the distribution of $D_{A_e} - D_{A_s}$ would have a mean of 0 and variance of $E(E+1)(2E+1)/3$. Figure 1 plots the frequency distribution of $D_{A_e} - D_{A_s}$ over the range of $-E \leq D_{A_e} - D_{A_s} \leq E$, for different values of E. But if $t < E$ due to t_e and t_s lying in same Epoch boundary, the second RDTSC (t_e) is moved to the consecutive Epoch [15]. Thus if the adversary repeats the pseudo-code several times and takes an average (denoted as T_{avg}) over the Timewarp timings, then for $t < E$ the Timewarp difference T_{avg} equals an E. on the other hand, for $t > E$, the randomness due to the difference

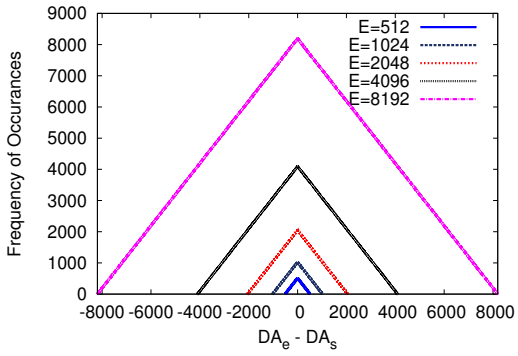

Figure 1: Distribution of difference of two random numbers D_{A_e} and D_{A_s} where $D_{A_e} - D_{A_s} \in (-E, E)$ and for various E's

of the apparent delay $D_{A_e} - D_{A_s}$ would be eliminated and T_{avg} would have a form as follows:

$$T_{avg} = \begin{cases} E & \text{if } t_e - t_s < E \\ (\lfloor \frac{t_e}{E} \rfloor E - \lfloor \frac{t_s}{E} \rfloor E) & \text{if } t_e - t_s > E \end{cases} \quad (6)$$

Thus from the equation, it shows that any time difference less than E cannot be measured accurately. On the contrary we claim that if the event times are larger compared to the epoch-length (i.e., $t > E$) then the Timewarp event times would be close to the actual event times.

We prove the above claim mathematically as follows: A timestamp counter value t can be represented as a sum of two values - an integral multiple of an Epoch(E) and an offset. Let the non-fuzzed timestamp counter value at the start of an event t_s be represented as

$$t_s = \lambda_s \times E + r_s \quad (7)$$

$\lambda_s = \lfloor t_s/E \rfloor$ where s is a non-negative integer and r_s which is the offset and is uniformly distributed in $[0, E)$. This is obtained from the remainder theorem, where t_s is the dividend, E the divisor, λ_s being the quotient, r_s the remainder. Similarly, the non-fuzzied time at the end of the event (i.e. t_e) can be written as

$$t_e = \lambda_e \times E + r_e, \quad (8)$$

where $\lambda_e = \lfloor t_e/E \rfloor$ is a non-negative integer and r_e being the offset is uniformly distributed in $[0, E)$. So the time taken for the event

$$t = t_e - t_s = (\lambda_e E - \lambda_s E) + (r_e - r_s). \quad (9)$$

Since both r_e and r_s are uniformly distributed over $[0, E)$, the average of $r_e - r_s$ over large samples is 0 as seen in Figure 1. Thus on averaging,

$$t_e - t_s = \lambda_e E - \lambda_s E \quad (10)$$

By incorporating the t_s and t_e from Equation 7 and 8 in the time fuzzied Equation 6 for T_{avg} we get when $t_e - t_s > E$

$$T_{avg} = \lambda_e E - \lambda_s E. \quad (11)$$

Thus,

$$T_{avg} = \begin{cases} E & \text{if } \lambda_e = \lambda_s \\ t_e - t_s & \text{if } \lambda_e > \lambda_s \end{cases} \quad (12)$$

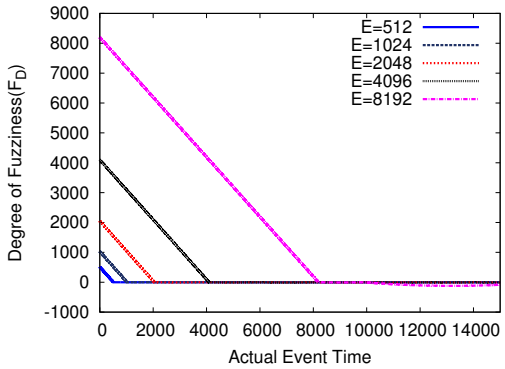

Figure 2: Difference of Timewarp encryption time and actual encryption time varies as the actual encryption time increases for various epoch sizes

Figure 2 depicts the amount of fuzziness that is injected by the Timewarp mechanism. The x-axis has the actual time for the event, while y-axis has degree of fuzziness (F_D) which we define as follows:

$$F_D = T_{avg} - (t_e - t_s) \tag{13}$$

We take the epoch size as $2^{13} = 8192$ cycles as was considered in [15] and different event times in the figure. Figure 2 shows that if event time $t_e - t_s$ is much lesser than the epoch time, the amount of fuzziness injected into the timing is significant. As the event time increases, the amount of fuzziness reduces and becomes almost 0 when t is greater than the epoch size. Based on this observation we can make the following conclusions:

- Timewarp is efficient at fuzzing time when $t << E$
- Timewarp is unable to fuzzy time when $t > E$.

In the next section we present an attack strategy using this observation.

4. SIDE-CHANNEL TIMING ATTACKS WITH TIMEWARP

Timing based side-channel attacks use the time for execution of an event as a means to gather secret information from a secure computation. Timing attacks work by distinguishing events based on their execution times. For example, certain cache based attacks distinguish between a cache hit and a miss based on the memory access time. In this section we propose a technique to show how events can be distinguished in spite of the Timewarp fuzziness applied. In the later part of the section we apply the technique on cache attacks against block ciphers.

4.1 Distinguishing Events Accurately with Timewarp

Consider an operation (denote Operation1) which could result in different timings based on events that occur. For example, the operation can be a memory access and the events a cache hit or a cache miss. We want to distinguish these events from their timing. If the time for Operation1 is $t_e - t_s > E$, then as seen in Section 3, the events can easily be distinguished as Timewarp is ineffective in this region. However, if $t_e - t_s << E$, then it is difficult to distinguish the events. We propose the following procedure in such cases.

Consider another operation (denoted Operation2) which takes a constant time, and is considerably longer than an epoch. We obtain the average time for Operation1 followed by Operation2 as shown below.

```
Ts = twRDTSC()
Operation1
Operation2
Te = twRDTSC()
T = Te - Ts
return T
```

Let T_{avg} be the average time for T. It has a value which is considerably larger than the epoch time E and therefore can be determined accurately as seen in Figure 2. Since the time for Operation2 is constant, variations in the time for Operation1 will be reflected in T_{avg}.

There are several ways in which Operation2 can be implemented. One way, which we found works well for cache attacks, is to repeat Operation1 say N times, where N is a positive integer chosen such that the cumulative time for all operations exceeds the epoch time. The pseudo code is as shown below:

```
Ts = twRDTSC()
For N number of times
    repeat Operation1
Te = twRDTSC()
T = Te - Ts
return T
```

We found that two cases arise:

- Either all N executions of Operation1 takes the same time. In which case the difference in time for the events gets amplified in T_{avg}.
- The first execution of Operation1 takes a variable time (depending on the event that occurred), while the remaining $N - 1$ executions take a constant time. The timing variations due to the first execution is then reflected in the total average time T_{avg}.

We use this method to mount cache attacks on block cipher with the Timewarp countermeasure in place. The next part of the section demonstrates these results.

4.2 Access-Driven Attacks with Timewarp Measurements

In the prime+probe cache access attacks [18, 25] the adversary tries to determine the cache sets that are modified by a cipher while it processes its input. Thus the attacker essentially understands which elements of a lookup table are accessed by the cipher. This methodology uses a spy program to interact with the microprocessor. Primarily, the spy fills the cache memory with its data. Then the cipher process is allowed to execute and after completion, the spy process executes again. This time the spy accesses its data again and measures the time of the memory accesses. A longer access time implies a cache miss, which in turn implies that the cache set was also used by the cipher. A short access time implies a cache hit, and the cache set was not accessed by the cipher. With the knowledge of the cache

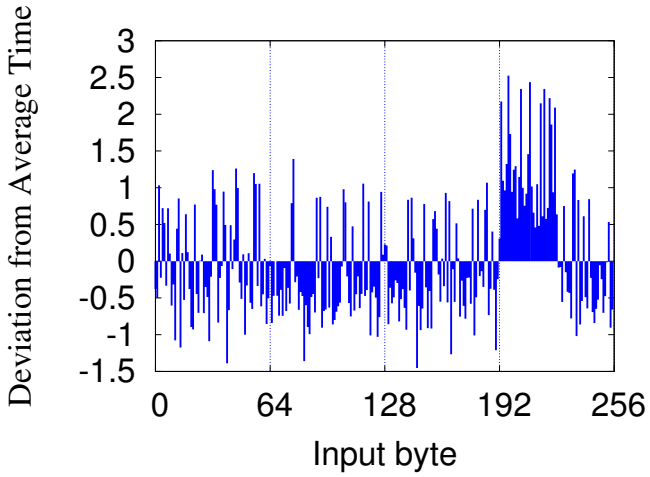

Figure 3: Timing Profiles for Key Byte for CLEFIA Implementation on Intel Xeon E5345. The x-axis has the possible values of the plaintext byte p_0 and the y-axis has the deviation of the encryption time from the average

sets accessed by the cipher, it is relatively easy to retrieve parts of the cipher's secret key.

Generally the difference between a cache hit and a miss is small, therefore with Timewarp indistinguishable. However, by applying the strategy of Section 4.2, during the second run, the spy reads each location N times, therefore is able to distinguish between a hit and a miss. We present the experimental results in Section 5.1.

4.3 Time-Driven Attacks with Timewarp Measurements

Unlike access-driven attacks where the attacker determines the parts of the lookup table accessed by the cipher, time-driven attack utilize differences in the encryption time. This is a much larger restriction from the adversary's perspective. In this section we show the applicability of time-driven attacks with fuzzied time. We consider a particular form of time-driven attacks known as profiled attacks. These attacks were first introduced by Bernstein in 2005 for AES [5], and then modified, enhanced, and applied for other ciphers in [9, 17, 21, 22, 23].

In a profiled time-driven cache attack the adversary first uses a known key to build a *timing profile* called *template* for each byte in the key. Figure 3 shows a typical timing profile. It has on the x-axis all possible values of a plaintext byte, while on the y-axis, the average execution time for a value of the byte. During the *attack phase*, a similar timing profile is built, only this time for an unknown key. The two timing profiles are then analyzed using Pearson's correlation to deduce the secret key.

The basis for the attack to work is that for any two execution times on the template profile say t_i and t_j, where $0 \leq i, j \leq 255$, there exists two execution times ($t_{i \oplus k}$ and $t_{j \oplus k}$, where $0 \leq k \leq 255$) on the unknown key timing profile such that if $t_i \preceq t_j$ then $t_{i \oplus k} \preceq t_{j \oplus k}$, where $\preceq$ is one of the following relations $\leq$, or $\geq$. The attack works only if we can accurately determine their relationships using the Timewarp measurements. If the execution time of the cipher is greater than the epoch time, then the timing can be accurately determined and such relationships in the timing can be built. However, if the execution time is less than the epoch time, then methods such as repeating encryptions (as seen in Section 4.1) have to be adopted to determine the required relationships. We have experimentally seen that such relationships can be found even when Timewarp fuzzing is in place. The following section has the results.

Table 1: Cache Memory Configurations for Evaluation Platforms

Type	Size	Line	Associativity
Intel Xeon (E5606)			
L1-Data	32KB	64byte	8-way
L1-Instruction	32KB	64byte	4-way
L2-Unified	256KB	64byte	8-way
L3-Unified	8MB	64byte	16-way
Intel Xeon (E5410)			
L1-Data	32KB	64byte	8-way
L1-Instruction	32KB	64byte	8-way
L2-Unified	6MB	4KB	24-way
AMD Opteron(TM) Processor 6272			
L1-Data	16KB	64byte	4-way
L1-Instruction	64KB	64byte	2-way
L2-Unified	2MB	64byte	16-way
L3-Unified	6MB	64byte	64-way
Intel Core 2 Duo (E8400)			
L1-Data	32KB	64byte	8-way
L1-Instruction	32KB	64byte	8-way
L2-Unified	6MB	64byte	24-way

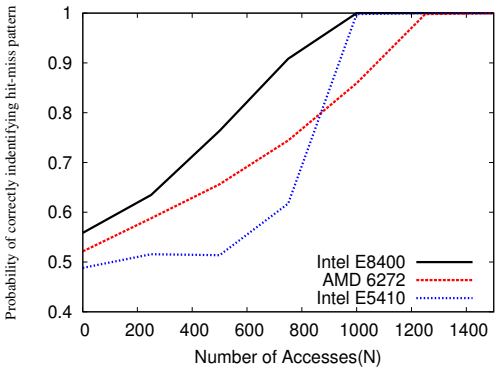

Figure 4: Probability of detecting a M or H correctly with different values of N

5. EXPERIMENTAL RESULTS

The attack strategy presented in the previous section works on both access-driven and time-driven attacks. We have done the experiments on an Intel Core 2 Duo E8400, AMD Opteron 6272, Intel Xeon E5410 and Intel Xeon E5606. The cache configuration of the platforms in shown in Table 1. In order to obtain the fuzzied time, we provided a wrapper (called twRDTSC) in software that read the RDTSC timestamp value and fuzzied it using Algorithm 1. This modification was encapsulated in a function called twRDTSC, and used in the attacks.

5.1 Access-Driven Cache Attacks

The motivation of our experiments was to determine how accurately Timewarp time measurements can distinguish between a cache hit and a miss. To do so, we define two arrays: $R1$ of size 32KB, which is as large as the L1 data cache, and acts as the spy buffer as described in Section 4.2, and $R2$ of size 2KB, which is accessed at random locations mimicking the memory accesses made by the cipher. This simulates a lookup table in a block cipher execution. Following is the pseudo code used in our simulations:

```
Access R1 to fill entire L1 data cache
Repeat L times{
  Access M elements of R2 at random locations
  For each memory block in R1{
      twRDTSC()
      Access an element in the memory block N times
      twRDTSC()
  }
  Compute the average of L iterations
}
```

The first set of accesses to $R1$ fills the entire L1 data cache. Some of this data is evicted when $R2$ is accessed. A typical cache access adversary, requires to find these blocks of data that are evicted. During the second set of accesses to $R2$, there are two possible events that can occur:

EM : If the memory block of $R1$ is evicted by $R2$, then the N accesses to the memory block of $R1$ would result in a cache miss followed by $N-1$ cache hits. This would look as follows: MHHHHHHH $\cdots$, where a cache miss is denoted by M while H denotes a cache hit.

EH : If the memory block of $R1$ is not evicted then the N accesses to $R1$ would result in all cache hits, which would look as follows: HHHHHH $\cdots$.

The value of N is chosen in such a way so that the time exceeds that of an epoch. Thus, the time can be accurately determined and used to distinguish between the events EM and EH. In our experiments on Intel Core 2 Duo E8400, AMD Opteron 6272 and Intel Xeon E5410 we kept L as 8, M as 512 and tested with various values of N. The time taken for the N accesses to a memory block depends on the value of N, and whether EM or EH is executed. For a given N, based on the non-fuzzied time, a threshold is first obtained. The fuzzied times were then classified according to this threshold. Figure 4 shows the probability that the fuzzied result matched the non-fuzzied result as the value of N increases. That is, for each of the L iterations and each of the memory blocks, a match is obtained if the non-fuzzied time and the fuzzied time detects the same event X, where X can be either M or H.

If the fuzziness is effective, then the probability of a match would be half. Implying that a miss or a hit is successfully hidden from the adversary. On the other hand if the fuzziness is ineffective, then the probability of a match would be greater than half. A probability of 1 indicates that it is as easy to distinguish between a M and a H with Timewarp as it is without. From Figure 4, we see that the probability is around $1/2$ when $N = 1$, and increases to 1 as N increases to 1000.

5.2 Profiled Time-Driven Attack on CLEFIA

In the previous subsection, an implementation of cache access attack was shown. In this section, we use the same idea to mount a profiled cache timing attack against the block cipher CLEFIA [22, 23]. The cipher is developed by Sony and is the current standard for light-weight cryptography. To apply the attack with Timewarp, the timing profiles were built by timing N encryptions of CLEFIA for the same plaintext. For the Intel Xeon E5606 and Intel Xeon E5410, N was found to be 6. Thus, for the modified attack, timing profiles were built with the known key and the unknown key by timing 6 encryptions of CLEFIA instead of 1. The profiles were then correlated to obtain a ranking of the keys in terms of the most likely to the least likely. The attack was termed a success if the correct key is one among the top five most likely.

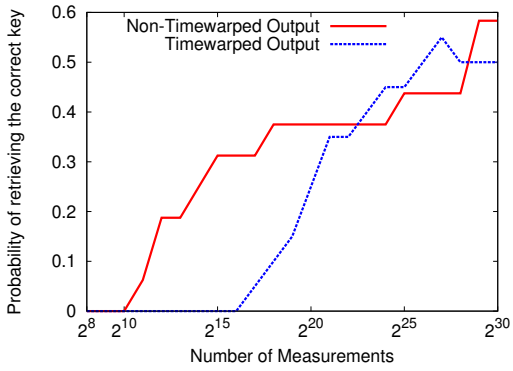

Figure 5: Increase in success rate of retrieving the key for CLEFIA as the number of iterations increases in power of 2's on Intel E5606

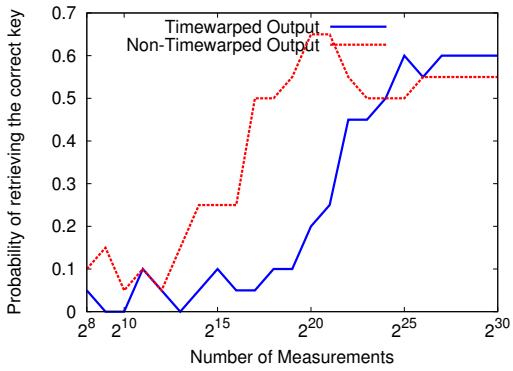

Figure 6: Increase in success rate of retrieving the key for CLEFIA as the number of iterations increases in power of 2's on Intel E5410

We repeated the attack varying the number of measurements from 2^9 to 2^{30}. The probability of getting the key is the number of times the attack is successful. We have repeated the experiments for 480 times and the probability is averaged over all the trials. For each of these cases an attack was also mounted with non-fuzzied timing measurements. Figure 5 and 6 shows the success rate of the attack on the Intel Xeon E5606 and Intel Xeon E5410 platform respectively. From the figures we can see that the probability

of obtaining keys without the Timewarp measurements is only slightly higher than the attack with Timewarp measurements being present. Thus showing proposed strategy is able to counter the Timewarp countermeasure.

6. CONCLUSIONS

Timewarp [15] proposes a minor modification to existing hardware architectures to fuzz the timestamp returned by the RDTSC instruction. The paper proposed to counter Timewarp by using the fact that the scheme is only effective when the time to be measured is less than the epoch time, and that the fuzziness introduced is limited by the size of the epoch. Thus showing that it is extremely complex to secure existing systems after an attack is surfaced. Hence security should not be an after thought and systems should be properly designed with security as an important criteria upfront.

7. REFERENCES

[1] M. Abe, editor. *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*. Springer, 2006.

[2] O. Aciiçmez, B. B. Brumley, and P. Grabher. New Results on Instruction Cache Attacks. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2010.

[3] O. Aciiçmez, Çetin Kaya Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In Abe [1], pages 225–242.

[4] O. Aciiçmez, W. Schindler, and Çetin Kaya Koç. Cache Based Remote Timing Attack on the AES. In Abe [1], pages 271–286.

[5] D. J. Bernstein. Cache-timing Attacks on AES. Technical report, 2005.

[6] E. Biham. A Fast New DES Implementation in Software. In E. Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[7] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke. Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs. In J. Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2010.

[8] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. *Computer Networks*, 48(5):701–716, 2005.

[9] A. Canteaut, C. Lauradoux, and A. Seznec. Understanding Cache Attacks. Research Report RR-5881, INRIA, 2006.

[10] L. Domnitser, A. Jaleel, J. Loew, N. B. Abu-Ghazaleh, and D. Ponomarev. Non-monopolizable caches: Low-complexity Mitigation of Cache Side-Channel Attacks. *TACO*, 8(4):35, 2012.

[11] D. Gullasch, E. Bangerter, and S. Krenn. Cache Games - Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy*, pages 490–505. IEEE Computer Society, 2011.

[12] D. Jayasinghe, J. Fernando, R. Herath, and R. Ragel. Remote cache timing attack on advanced encryption standard and countermeasures. In *ICIAF*, pages 177–182, 2010.

[13] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. *J. Comput. Secur.*, 8(2,3):141–158, 2000.

[14] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, London, UK, 1996. Springer-Verlag.

[15] R. Martin, J. Demme, and S. Sethumadhavan. TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-Channel Attacks. In *ISCA*. IEEE, 2012.

[16] M. Neve and J.-P. Seifert. Advances on Access-Driven Cache Attacks on AES. In E. Biham and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2006.

[17] M. Neve, J.-P. Seifert, and Z. Wang. A Refined Look at Bernstein's AES Side-Channel Analysis. In F.-C. Lin, D.-T. Lee, B.-S. Lin, S. Shieh, and S. Jajodia, editors, *ASIACCS*, page 369. ACM, 2006.

[18] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In D. Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[19] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel, 2002.

[20] C. Percival. Cache Missing for Fun and Profit. In *Proc. of BSDCan 2005*, 2005.

[21] C. Rebeiro, M. Mondal, and D. Mukhopadhyay. Pinpointing Cache Timing Attacks on AES. In *VLSI Design*, pages 306–311. IEEE Computer Society, 2010.

[22] C. Rebeiro and D. Mukhopadhay. Boosting Profiled Cache Timing Attacks with Apriori Analysis. *Information Forensics and Security, IEEE Transactions on*, PP(99):1, 2012.

[23] C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga. Cache Timing Attacks on CLEFIA. In B. Roy and N. Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2009.

[24] Shay Gueron. Intel Advanced Encryption Standard (AES) Instructions Set (Rev : 3.0), 2010.

[25] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 23(2):37–71, 2010.

[26] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In C. D. Walter, Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2003.

[27] Y. Tsunoo, E. Tsujihara, M. Shigeri, H. Kubo, and K. Minematsu. Improving Cache Attacks by Considering Cipher Structure. *Int. J. Inf. Sec.*,

5(3):166–176, 2006.

[28] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In D. M. Tullsen and B. Calder, editors, *ISCA*, pages 494–505. ACM, 2007.